

MPI Version of the Serial Code with Two Dimensional Decomposition

**Luke Lonergan
High Performance Technologies, Inc.**

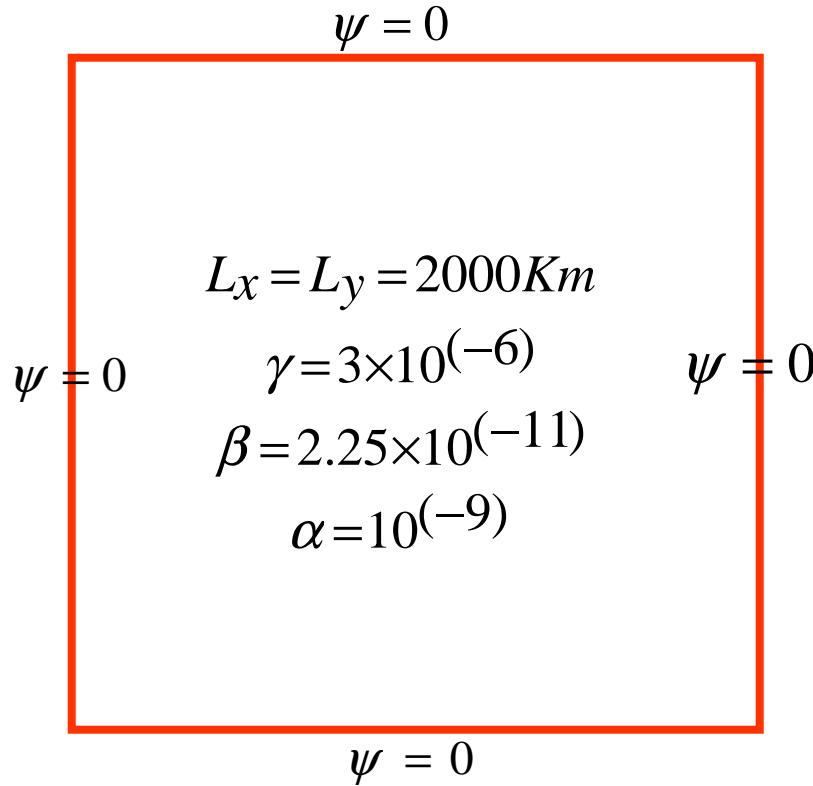
Strategy

- Step1: Setup the MPI environment in the code
- Step2: Domain decomposition
- Step3: Introduce “Virtual Cartesian topology”
- Step4: Modify “forcing”
- Step5: Modify boundary conditions
- Step6: Modify “residual”
- Step7: write exchange.f
- Step8: compile and run the code

STEP1: introduce the MPI environment

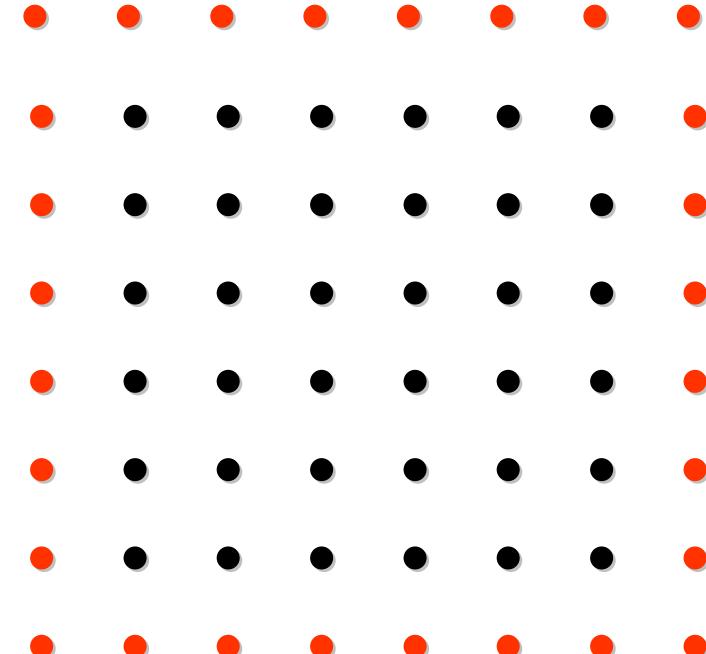
- cd /mpi_2d/step1
- edit/view the file step1.f : which has been
 - changed as follows
 - included the mpi header file (mpif.h)
 - declared integer variables size, rank, ierr
 - initialized MPI
 - determined process group
 - determined the rank of calling process
 - terminated MPI environment

Domain Discretization in the serial code



$$\gamma \left(\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} \right) + \beta \frac{\partial \psi}{\partial x} = f$$

$$f = -\alpha \sin\left(\frac{\pi y}{L_y}\right)$$

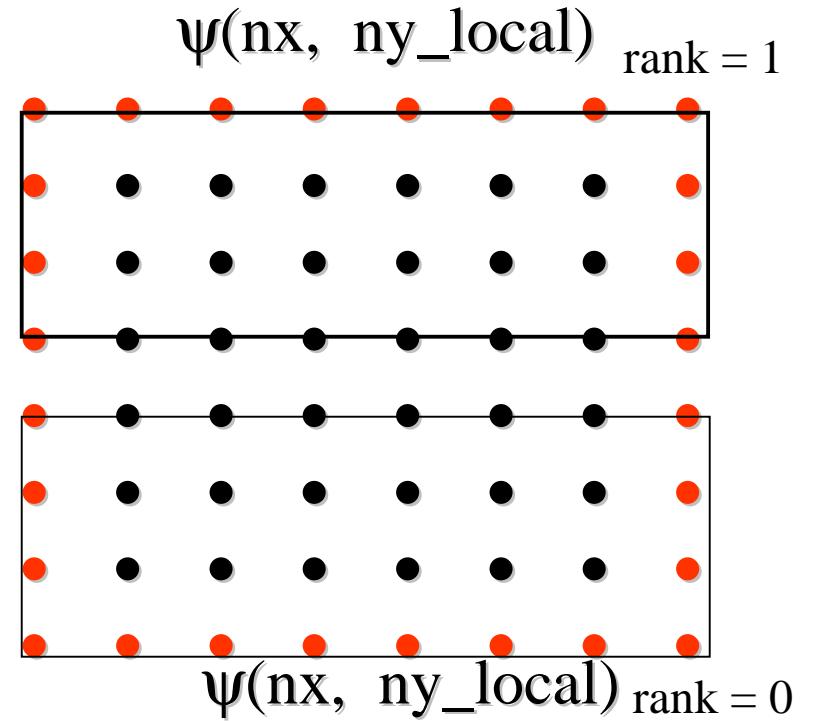
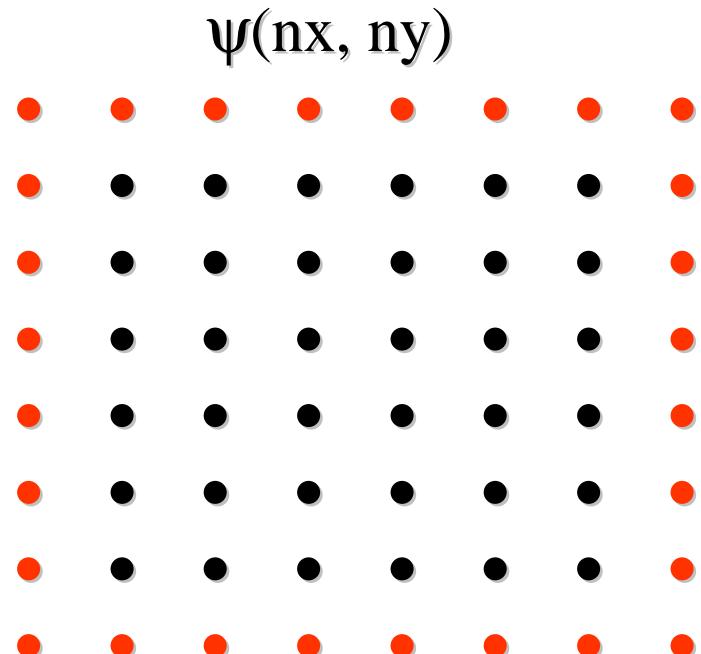


$$\begin{aligned} \psi_{i,j} &= a_1 \psi_{i+1,j} + a_2 \psi_{i-1,j} + \\ &\quad a_3 \psi_{i,j+1} + a_4 \psi_{i,j-1} - \\ &\quad a_5 f_{i,j} \end{aligned}$$

$$a_k = a_k(\gamma, \beta, \Delta x, \Delta y)$$

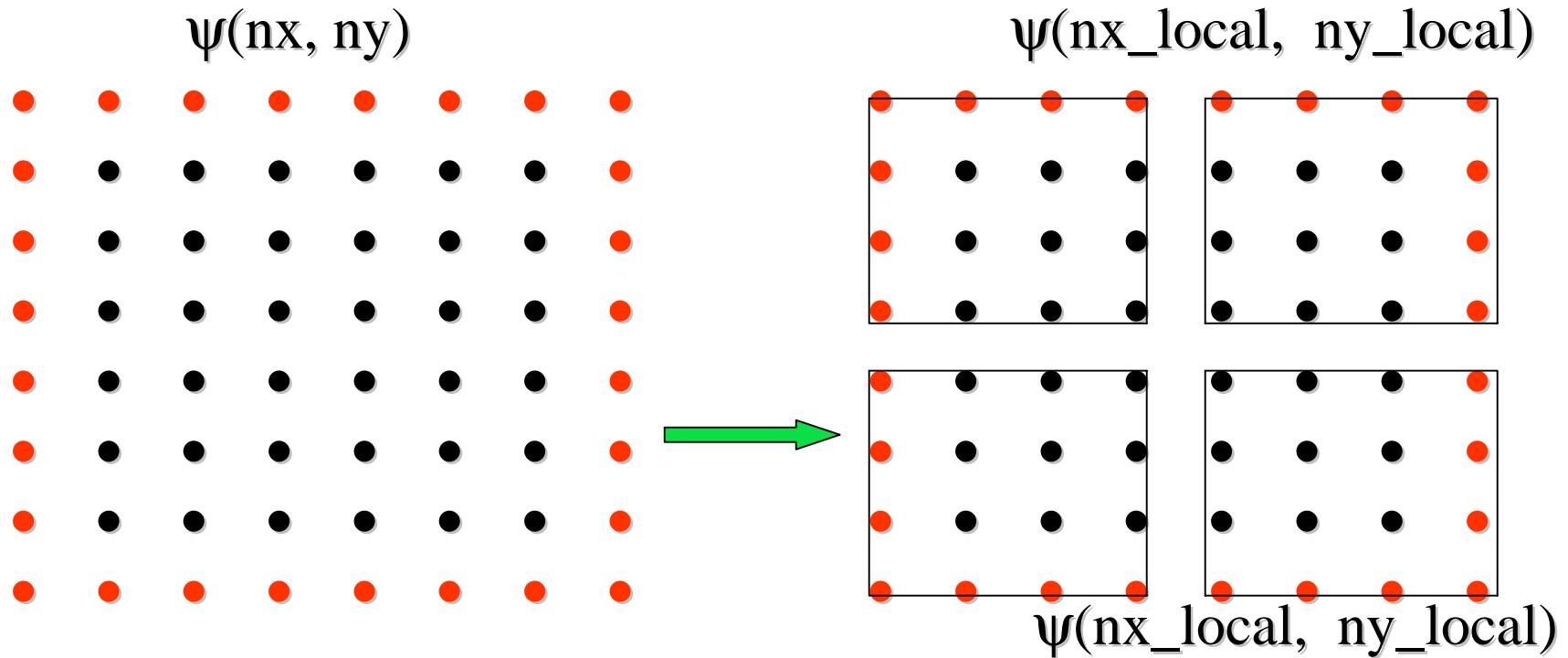
1D Decomposition

Physical domain is sliced into slabs so that computation in each slab will be handled by different processors.



2D Decomposition

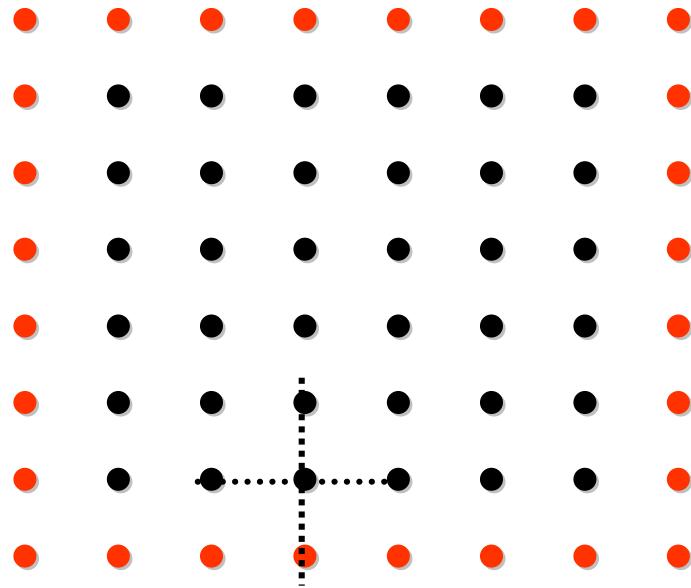
Physical domain is sliced into blocks so that computation in each block will be handled by different processors.



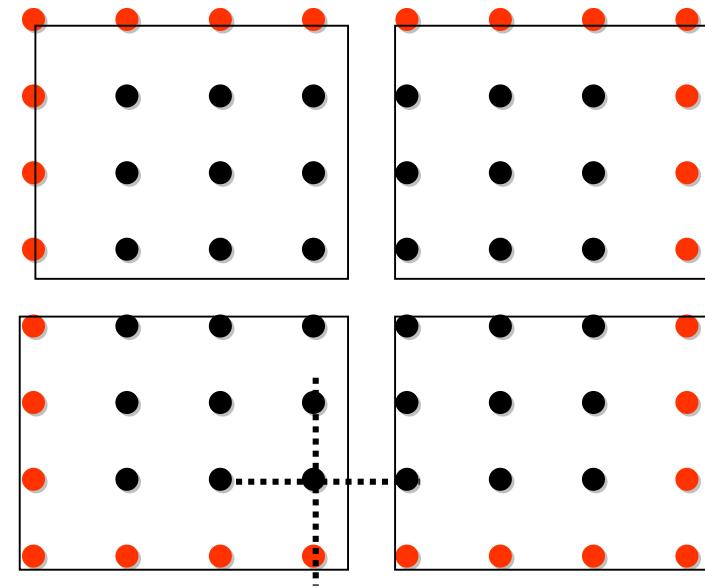
Jacobi Iteration on Decomposed Data

$$\psi_{\text{new}}(i, j) = a_1 * \psi(i+1, j) + a_2 * \psi(i-1, j) + a_3 * \psi(i, j+1) + a_4 * \psi(i, j-1) - a_5 * f(i, j)$$

$i=2, nx-1 ; j=2, ny-1$



$i=2, nx_local-1 ; j=2, ny_local-1$

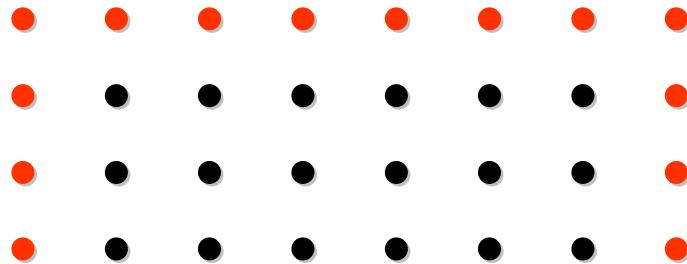


$\psi(nx, ny)$

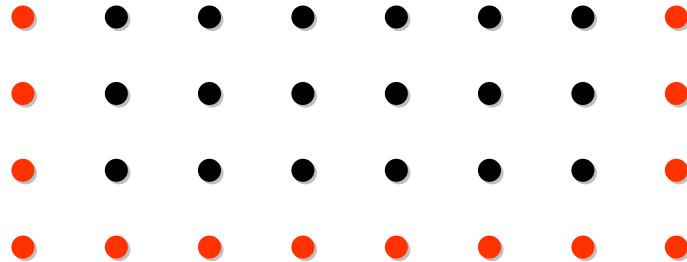
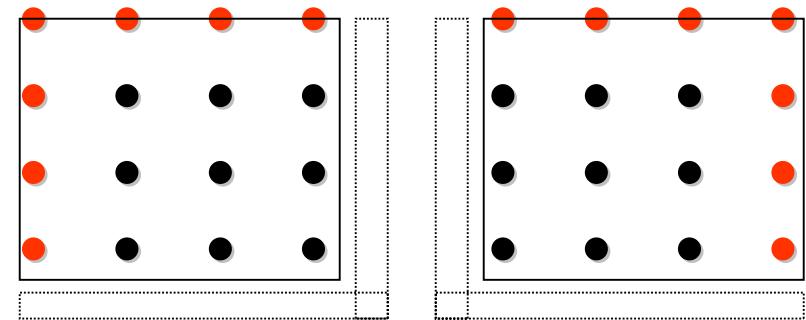
$\psi(nx_local, ny_local)$

“Ghost” Rows and Columns

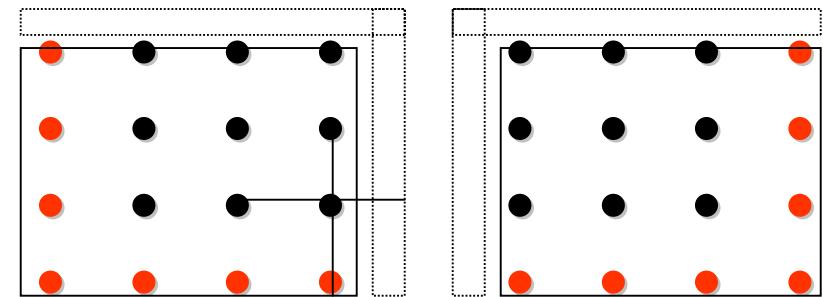
$i=2, nx-1 ; j=2, ny-1$



$i=2, mynx-1 ; j=2, myny-1$



$\psi(nx, ny)$



$\psi(mynx, myny)$

A choice for $mynx$ and $myny$

- $mynx = [(nx-2)/nprocx] + 2]$
- $myny = [(ny-2)/nprocy] + 2]$

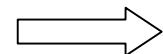
STEP2: 2d decomposition of the arrays in the serial code

- cd /mpi2d_step2
- edit/view step2.f(which has been changed as follows)
 - declared integer variable nprocs
 - declared integer variables **mynx** and **myny**
 - set parameter(**mynx** = [(nx-2)/nprocx] + 2])
 - set parameter(**myny** = [(ny-2)/nprocy] + 2])

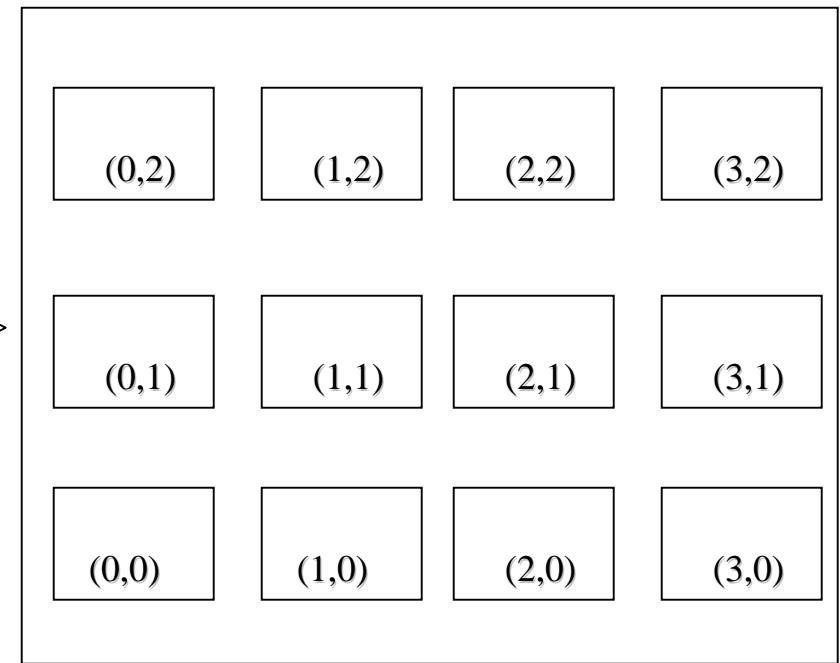
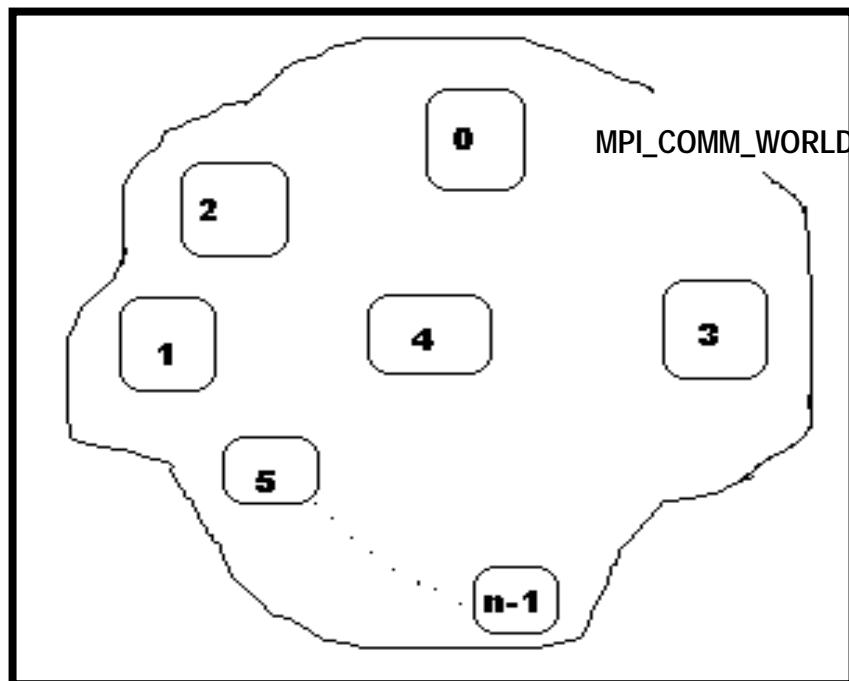
Virtual Cartesian Topology

Decomposition of processors in the natural coordinate (e.g. x, y coordinates)

MPI_COMM_WORLD



COMM_2D



comm_2d

MPI_COMM_WORLD \rightarrow **COMM_2D**

MPI_CART_CREATE(**MPI_COMM_WORLD**,
ndim, **dims**, **periods**, **reorder**, **comm_2d**, **ierr**)

ndim = 2

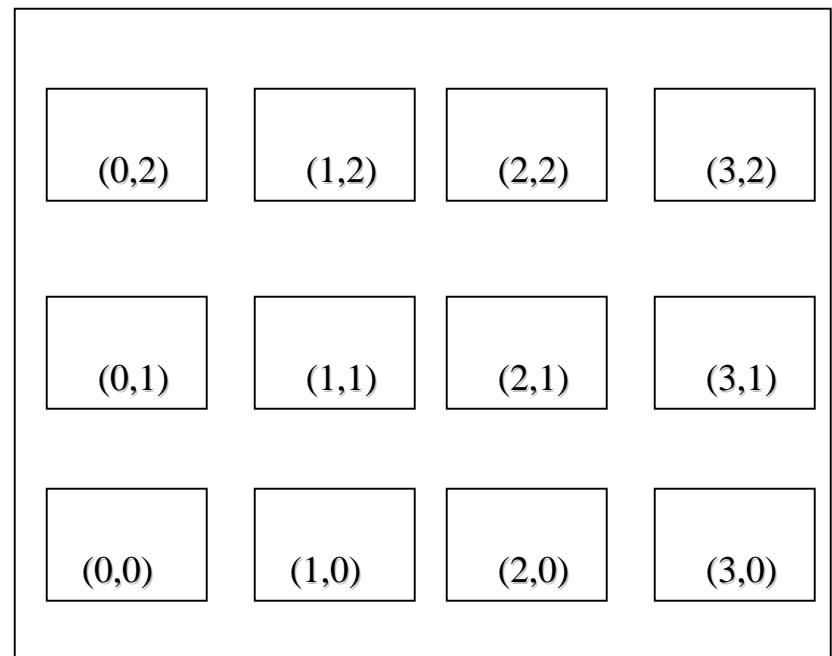
dims(1) = 4

dims(2) = 3

periods(1) = .false.

periods(2) = .false.

reorder = .true.

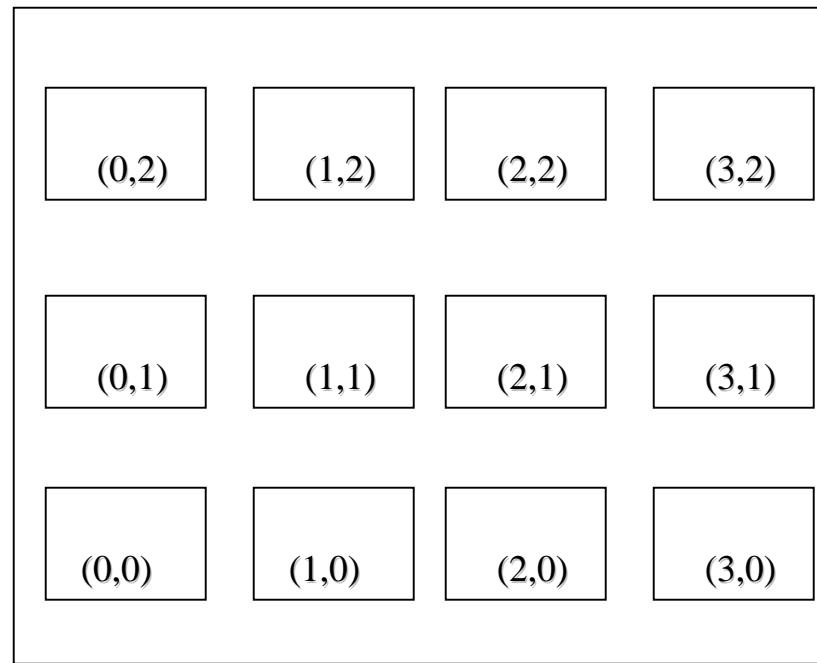


comm_2d

Coordinates of the Virtual Topology

integer coords(2)

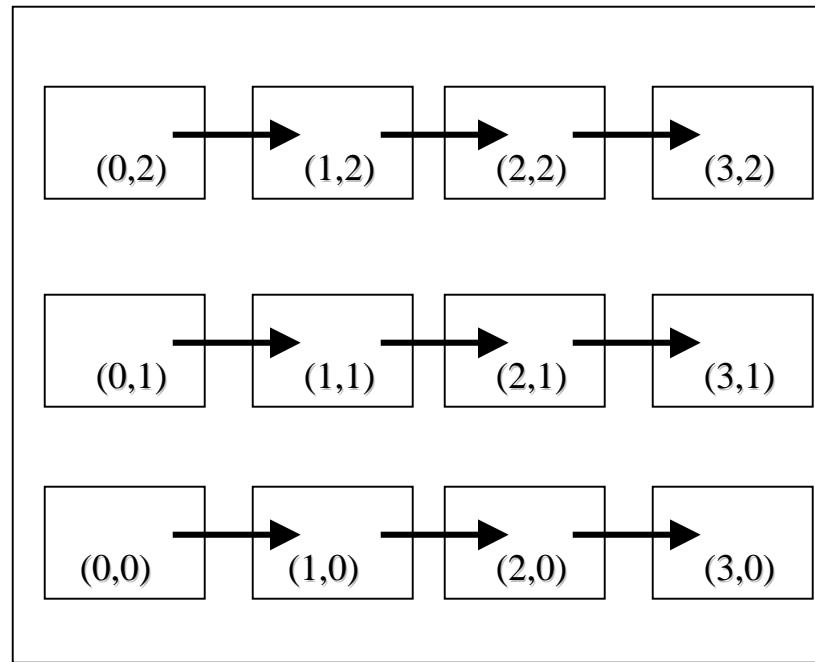
MPI_CART_GET (comm_2d, 2, dims,
periods, coords, ierr)



comm_2d

Neighbors on My Left and Right

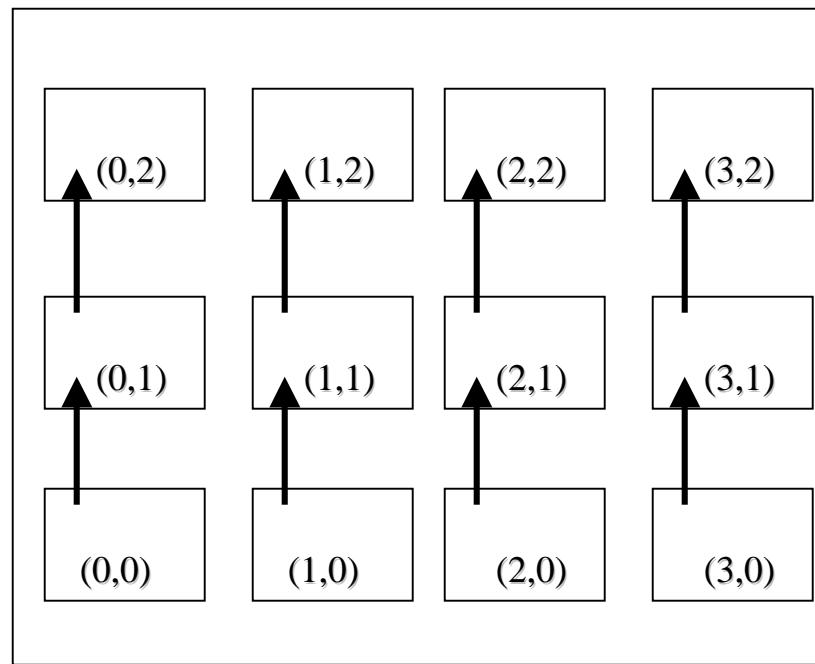
`MPI_CART_SHIFT(comm_2d, direction, shift,
first_dir, sec_dir, ierr)`



`MPI_CART_SHIFT(comm_2d, 0, 1, left, right, ierr)`

Neighbors Below and Above

`MPI_CART_SHIFT(comm_2d, direction, shift,
first_dir, sec_dir, ierr)`



`MPI_CART_SHIFT(comm_2d, 1, 1, down, up, ierr)`

STEP3 : Write a routine that sets up the 2d virtual topology, gets the coordinates and identifies all four neighbors

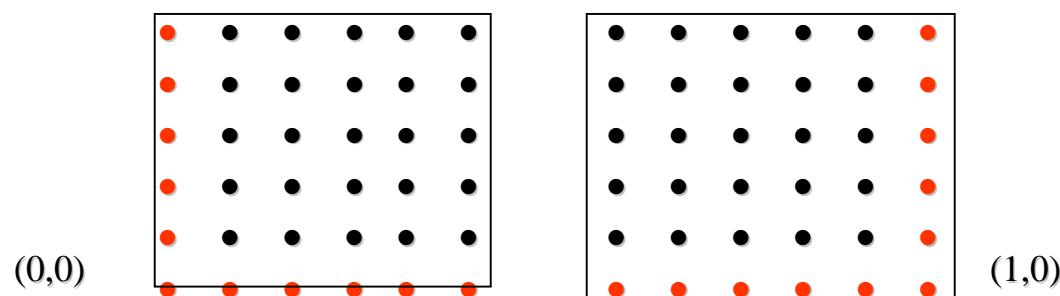
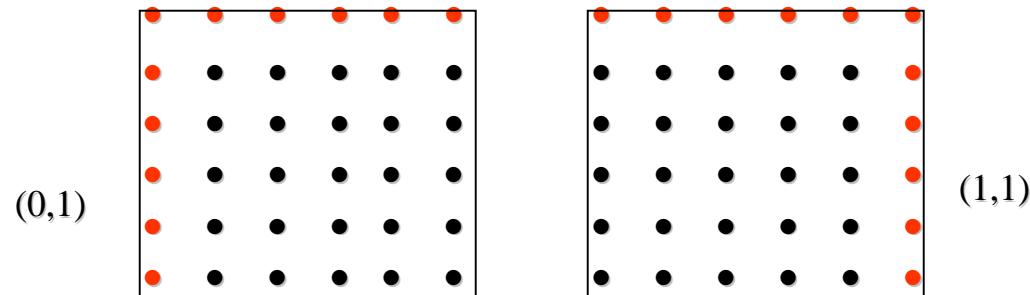
- cd /mpi_2d/step3
- edit/view the file: topo.f
- compile and run topo with np = 12
- edit/view the subroutine topology.f

Decomposing the Forcing Function

$$f(i,j) = -\alpha \sin(\pi * y(j)/L_y)$$

$$y_j = y_{\text{start}} + (j-1) * \Delta y$$

`ystart = coords(2)*(myny-2) *Δy`



STEP4: decomposing the forcing function

- cd /mpi_2d/step4
- edit/view the file forcing.f
 - includes coords as argument
 - ystart = coords(2) * (mny-2) * dy
 - yj = ystart + (j-1)* dy

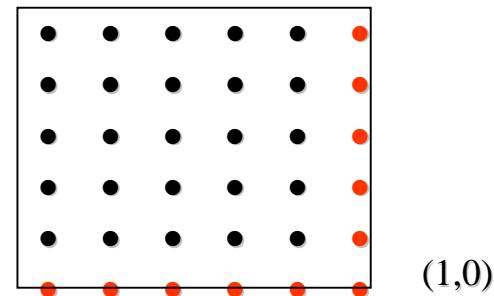
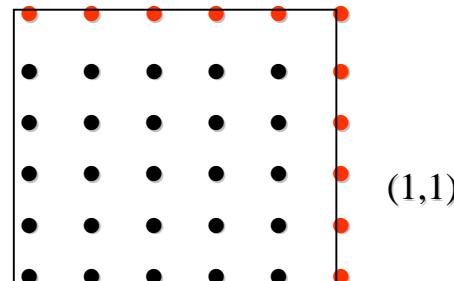
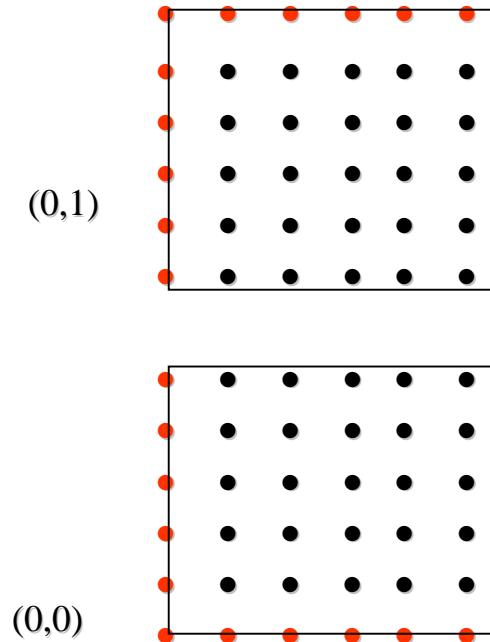
Decomposing the Boundary Conditions

If (coords(2) .eq. nprocy-1)

$\psi(i, myny) = 0 ; i = 1, mynx$

If
(coords(1)
.eq.
0)

$\psi(1, j) = 0 ;$
 $j = 1, myny$



If
(coords(1)
.eq.
nprocx-1)
 $\psi(1, j) = 0 ;$
 $j = 1, myny$

If (coords(2) .eq. 0)

$\psi(i, 1) = 0 ; i = 1, mynx$

STEP5: Decomposing the Boundary conditions

- cd mpi_2d/step5
- edit/view the file bcs.f

STEP6: Modifying the routine “residual”

- cd mpi_2d/step6
- edit/view the file residual.f

Exchanging the Ghost Data - (i)

Step 7a: edit/view exchange.f

call MPI_SEND(

(psi(1,myny-1),

mynx,

MPI_REAL,

up,

0,

comm_2d,

ierr)

MPI_RECV(

(psi(1, 1),

mynx,

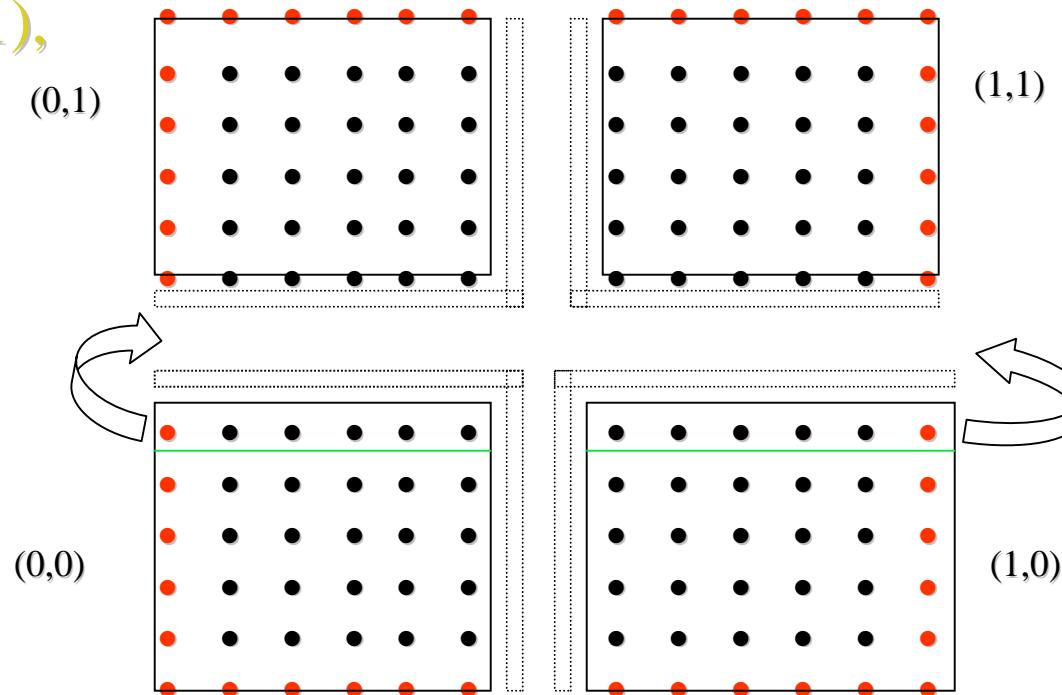
MPI_REAL,

down,

0,

comm_2d,

stat, ierr)



Exchanging the Ghost Data - (ii)

Step 7b: edit/view exchange.f

`MPI_RECV(`

`MPI_SEND(`

`(psi(1, 2),`

`mynx,`

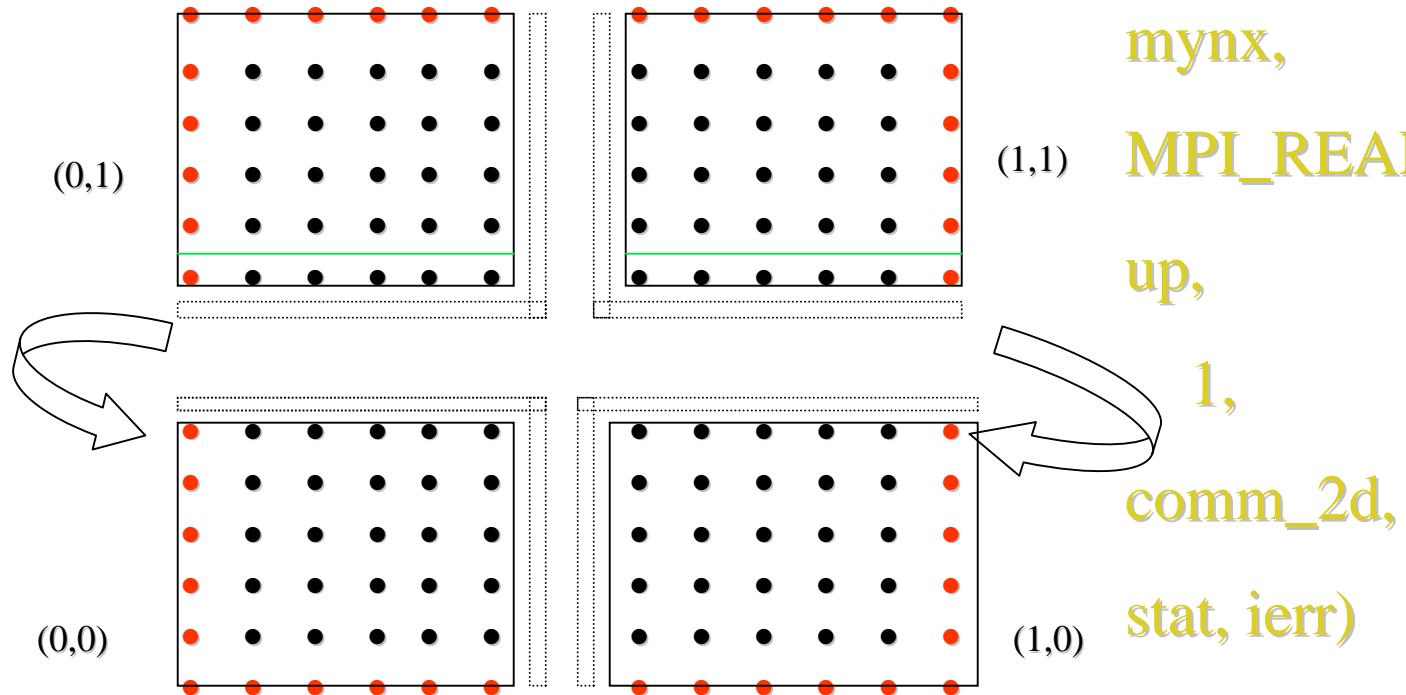
`MPI_REAL,`

`down,`

`1,`

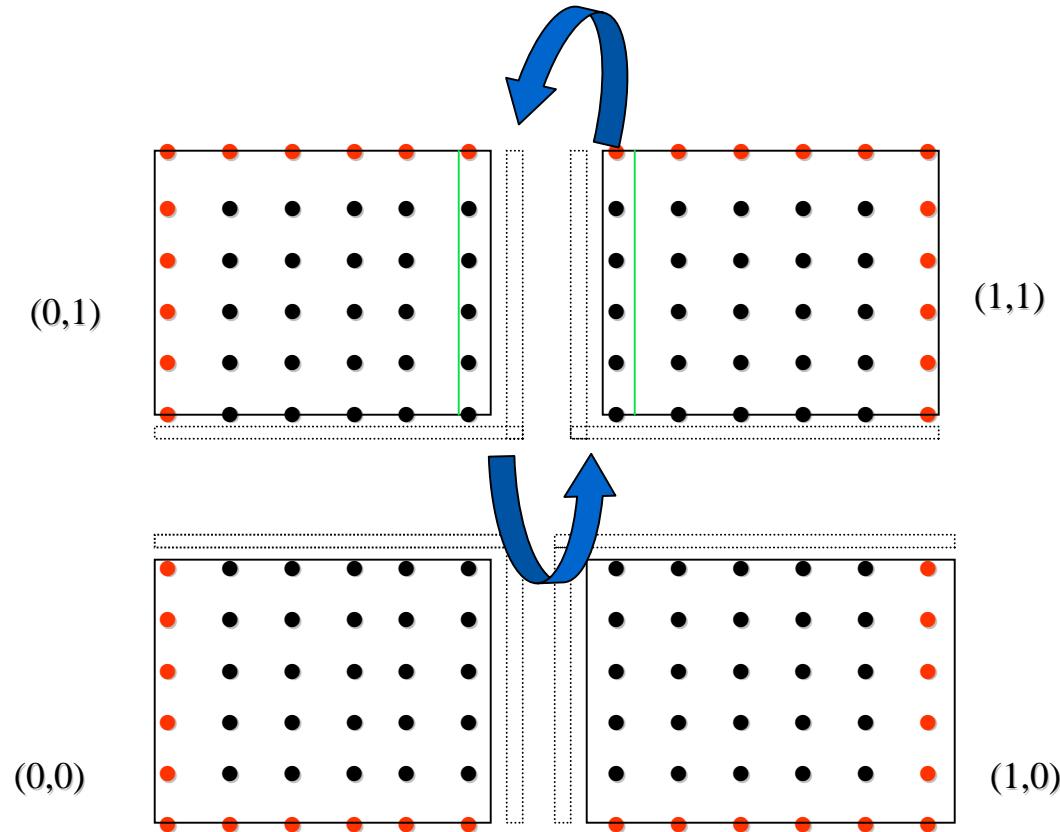
`comm_2d,`

`ierr)`



Exchanging the Ghost Data - (iii)

Step 7c,d:



Derived Datatypes

29	30	31	32	33	34	35
22	23	24	25	26	27	28
15	16	17	18	19	20	21
8	9	10	11	12	13	14
1	2	3	4	5	6	7

```
MPI_TYPE_VECTOR(5, 1, 7, MPI_REAL,  
newtype, ierr)
```

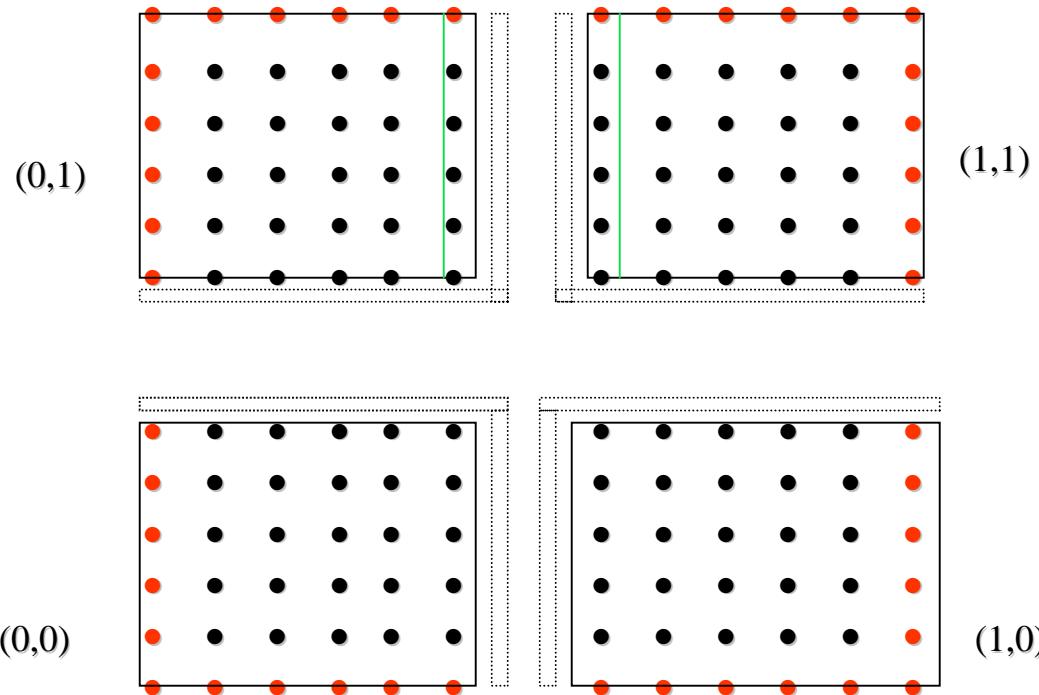
Derived Datatypes

- regular stride newtype

```
call MPI_TYPE_VECTOR(count,  
                      blocklength,  
                      stride,  
                      datatype,  
                      newtype  
                      ierr)  
call MPI_TYPE_COMMIT(newtype, ierr)  
call MPI_TYPE_FREE(newtype)
```

Exchanging the Ghost Data - (iii)

MPI_TYPE_VECTOR(myny ,1, mynx,MPI_REAL, strided, ierr)



Derived Datatypes

- regular strided

```
call MPI_TYPE_VECTOR(myny,
                      1,
                      mynx,
                      MPI_REAL,
                      strided,
                      ierr)
```

```
call MPI_TYPE_COMMIT(strided, ierr)
```

```
call MPI_TYPE_FREE(strided)
```

Exchanging the Ghost Data - (iii)

Step 7c: edit/view exchange.f

call MPI_SEND

(psi(mynx-1,1),

1,

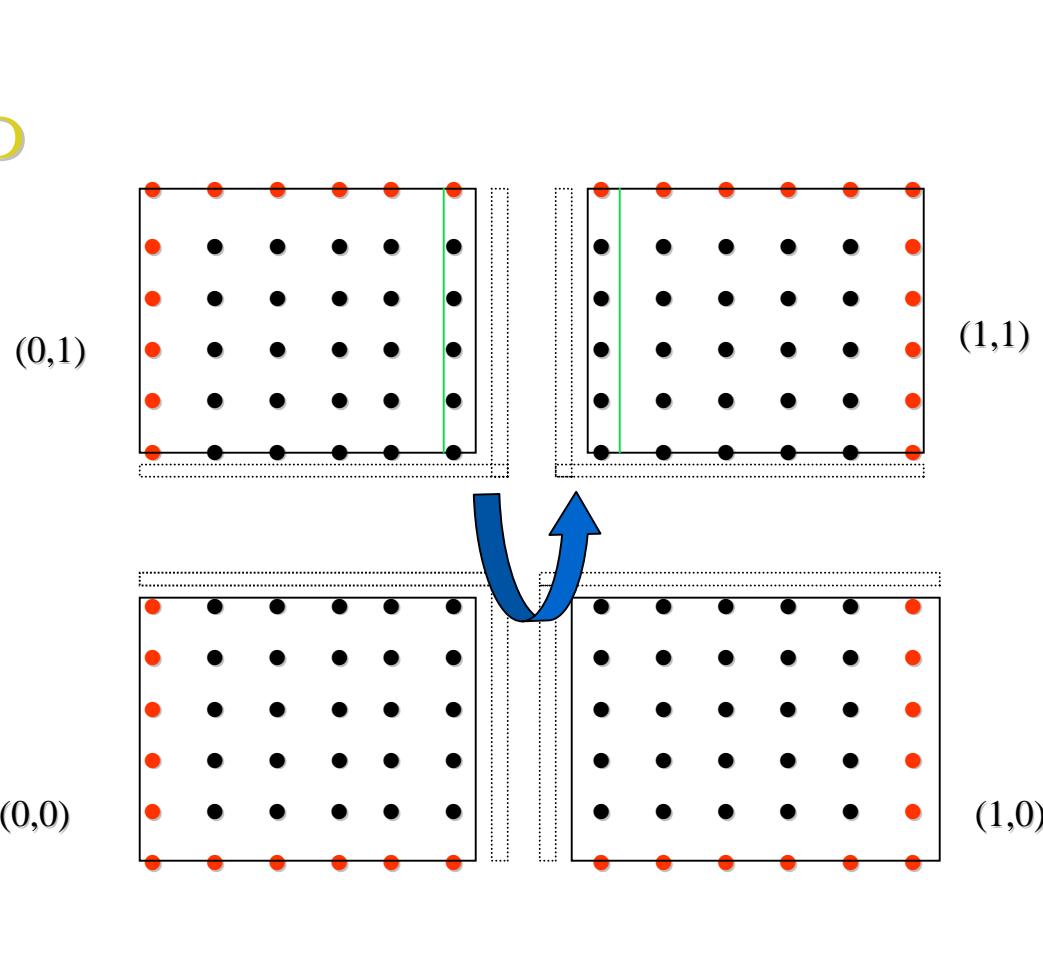
strided,

right,

2,

comm_2d,

ierr)



MPI_RECV

(psi(1,1),

1,

strided,

left,

2,

comm_2d,

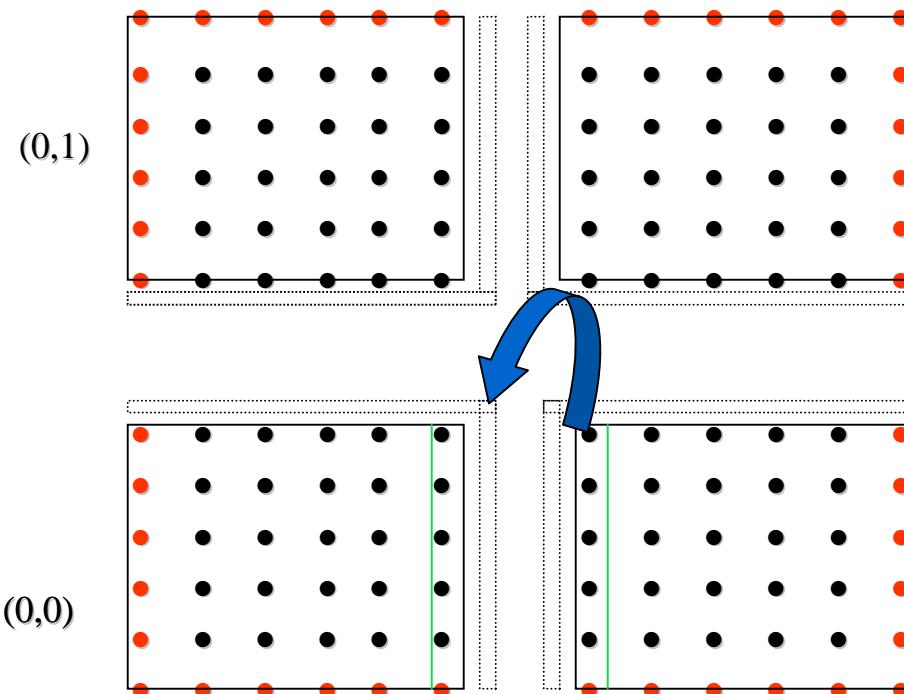
stat,

ierr)

Exchanging the Ghost Data - (iv)

Step 7d: edit/view exchange.f

```
call  
MPI_SEND(  
(psi(2,1),  
1,  
strided,  
left,  
3,  
comm_2d,  
ierr
```



```
                  MPI_RECV(  
(psi(mynx,1),  
1,  
strided,  
right,  
3,  
comm_2d,  
stat,  
ierr
```

STEP8: Compiling and running stml_mp2d

- cd mpi_2d/step8
- edit/view step8.f
 - MPI Virtual Topolgy Parameters
 - strided datatype is defined
 - routine topology is called
 - new datatype “strided” defined and committed
 - ‘strided’ and “comm_2d” are freed at the end

STEP8: Compiling and running step8.f

- compile and run step8.f with np = 4
- compare with the results of serial code
- rename step8.f stml_mpi2d.f